Introduction and Outline

The research proposed is to develop methodologies and tools for designing and implementing very large-scale real-time embedded computer systems that

- **achieve ultra high computational performance through use of parallel hardware architectures;**
- **achieve and maintain functional integrity via distributed, hierarchical monitoring and control;**
- **are required to be highly available; and**
- **are dynamically reconfigurable, maintainable, and evolvable.**

The specific application that will drive this research and provide a test platform for it is the trigger and data acquisition system for BTeV[1], an accelerator-based High Energy Physics (HEP) experiment to study matter-antimatter asymmetries (also known as Charge-Parity violation) in the decays of particles containing the bottom quark. BTeV was recently approved by Fermilab[2] and will be constructed over the next 5-6 years to run in conjunction with the Fermilab Tevatron Collider. The experiment is expected to run for at least 5 years. It requires a massively parallel, heterogeneous cluster of computing elements to reconstruct 15 million particle interactions (events) per second and uses the reconstructions to decide which events to retain for further data analysis. **Creating usable software for this type of real-time embedded system will require research into solutions of general problems in the fields of computer science and engineering. We plan to approach these problems in a way that is general, and to produce methodologies and tools that can be applied to many scientific and commercial problems.** During this project, the research results will be carried into the high-school system through projects which enhance existing infrastructure for attracting students into science and engineering disciplines.

The classes of systems targeted by this research include those embedded in environments, like BTeV, that produce very large streams of data which must be processed in real-time using data dependent computation strategies. Such systems are inextricably tied to the environment in which they must operate, and must perform complex computations within the timing constraints mandated by their environments. These systems require **ultra high performance** (on the order of 10^{12} operations per second). The level of performance requires **parallel hardware architectures,** which in the case of BTeV is composed of a mix of thousands of commodity processors, special purpose processors such as Digital Signal Processors (DSPs), and specialized hardware such as Field Programmable Gate Arrays (FPGAs), all connected by very high-speed networks. The systems must be **dynamically reconfigurable**, to allow a maximum amount of performance to be delivered from the available and potentially changing resources. The systems must be **highly available**, since the environments produce the data streams continuously over a long period of time, and interesting phenomena important to the analysis being done are rare and could occur in the data at any time. To achieve the high availability, the systems must be **fault tolerant**, **self-aware**, and **fault adaptive**, since any malfunction of processing elements, the interconnection switches, or the front-end sensors (which provide the input stream) can result in unrecoverable loss of data. Faults must be corrected in the shortest possible time, and corrected **semi-autonomously** (i.e. with as little human intervention as possible). Hence **distributed** and **hierarchical monitoring and control** are vital.

We believe that there are very significant advantages to connecting this research to the BTeV experiment. Not only will the software and methods produced by this research have significant impact on one of the most important areas of investigation in HEP, but the generalizable computer engineering research will also be directly applicable to a large class of similar real-time embedded computer systems. The BTeV trigger system hardware, which will be provided by Fermilab as part of the experiment, will supply an extremely important ingredient in this project: a large test-bed that represents millions of dollars of equipment and comes with a highly motivated set of users who will test the methodologies and tools developed in an extremely harsh environment over an extended period of time. The test-bed will be built gradually as the proposed research progresses, from a 5% system in 2002 to a full system in 2005-2006. It will therefore be possible for the software developers, aided and supported by the experimenters, to test and refine the software and strategies continuously and incrementally throughout the lifetime of this project. The close interdisciplinary contact between the experimenters and computer scientists will also help introduce important computer science research into the HEP community, which has not always been aware of work that has been done in this area and has not taken full advantage of it.

The team that has been assembled to carry out this research consists of the leaders of the BTeV trigger and data acquisition system development efforts and Computer Scientists who are experts in the field of embedded systems, real-time systems, and fault tolerant computing[3]. The Computer Scientists come from universities that are strongly involved in BTeV, and from Fermilab. The team is committed to carrying out the proposed R&D and implementing a series of systems of increasing size and complexity, using the experience gained at each stage to refine and improve the system until it is demonstrated to scale to the full BTeV system.

The outline of the proposal is as follows: Section 1 gives a brief description of the BTeV data acquisition and trigger system and explains issues relating to scalability, flexibility, dynamic reconfigurability, partitionabilty, and fault-tolerance. Section 2 describes the software required for systems of such complexity and our approach toward developing it. Section 3 reviews existing work in this area and discusses how the proposed research will adopt and extend it. Section 4 discusses those features that will make the research applicable to a wide range of scientific and information technology problems, along with examples of additional applications. Section 5 describes the educational outreach activities of the project. Section 6 describes the collaboration, its organization, and its personnel and lists project deliverables and milestones. Section 7 contains concluding remarks.

1. The BTeV Trigger System

This section has two parts. The first part describes the BTeV triggering and data acquisition system, in order to explain the problem that must be solved and the basic architecture and scale of the system that is planned to address it. The "trigger" or "event filter" algorithms that run on the resulting hardware platform are briefly described. These algorithms, which will largely be written by physicists from the BTeV experiment, are not, however, the thrust of the research proposed here. The second part of this section addresses the requirements of the infrastructure required to keep the trigger system operating, to assure that it is working correctly, and to detect and adapt to fault conditions both within the computing platform and within the experiment and machine environment. This has been called out[4] as the major challenge in implementing the BTeV trigger:

> "Regarding the robustness and integrity of the hardware and software design of the trigger system, these issues and concerns have only begun to be addressed at a conceptual level by BTeV proponents … Given the very complex nature of this system where thousands of events are simultaneously and asynchronously cooking, issues of data integrity, robustness, and monitoring are critically important and have the capacity to cripple a design if not dealt with at the outset. It is simply a fact of life that processors and processes die and get corrupted, sometimes in subtle ways. BTeV has allocated some resources for control and monitoring, but our assessment is that the current allocation of resources will be insufficient to supply the necessary level of "self-awareness" in the trigger system… Without an increased pool of design skills and experience to draw from and thermalize with, the project will remain at risk. The exciting challenge of designing and building a real life pixel-based trigger system certainly has the potential to attract additional strong groups."

The main thrust of the R&D proposed here is to address this key concern, which is a generic concern for highly-reliable embedded real-time systems of this scale and complexity

1.1 The BTeV Trigger System Hardware and Filtering Algorithms

BTeV is a High Energy Physics (HEP) project that will carry out an ambitious experiment to search for and study differences between the decay of particles containing b-quarks and the corresponding decays of anti-particles containing b-antiquarks. The study of this asymmetry will shed light on the preponderance of matter over antimatter in the observable universe and is one of the intense areas of research in elementary particle physics[5]. BTeV will utilize the Fermilab Tevatron Collider for this research. The Tevatron will produce 15 million high-energy particle collisions (also referred to as interactions) per second at the center of the BTeV detector. Each one of these interactions typically creates between 10 to 100 subatomic particles that will travel through the detector, where they will be tracked and identified. The interactions that are likely to exhibit b-quark asymmetries are expected to occur once for every 1 million collisions in the BTeV detector. This means that for each year of operation BTeV must "filter" data from over one hundred trillion interactions to select a sample of 10 billion b-quark decays from which a few million will be used to reveal the mysteries of matter-antimatter asymmetries.

BTeV will generate an enormous amount of data, about 1.5 terabytes per second. The factors that contribute to this exceptionally large data rate are the interaction rate (15 million collisions/s), the large number of particles that are produced in the collisions, and the large number of electronic sensor channels (30 million channels in the current design) in the detector. Because recording all of the data on an archival medium for later analysis is simply impossible, the challenge for the BTeV trigger system is to analyze data from the detector in real-time and to select interesting b-quark interactions to write to permanent storage for subsequent offline analysis.

Electronic trigger systems are common in HEP experiments[6]. However, BTeV is pursuing an ambitious trigger strategy that is unique in HEP. Most other experiments use a simple "first level" trigger, based on dedicated hardware which makes relatively unsophisticated decisions based on the most obvious, but not necessarily the most fundamental, differences between the signal and the background events. Typically, these triggers accept only very specific event topologies that conform to the idea of what is important physics at that moment. This reduces the number

of interactions that must be processed by subsequent trigger levels, but the simple first-level trigger limits physics analyses by rejecting potentially interesting data. For BTeV a conventional first-level trigger of this type would restrict the experiment to a limited selection of b-quark particle decays, and would thereby prevent us from pursuing the broad range of physics analyses, including some that are "off the beaten path", that we feel are needed to study b-quark asymmetries. Consequently, the BTeV trigger must be considerably more complex than triggers used for other experiments. The design of the trigger is driven primarily by the physics goals of the experiment.

The BTeV strategy is to trigger on the most fundamental property that differentiates particles containing the b-quark from other types of particles. That property is the presence of an interaction vertex, where the B particle, the anti-B particle, and many other particles are produced, followed by vertices a few hundred microns away where the B particles decay. Detecting such small vertex separations requires the reconstruction of all vertices using a complex pattern recognition algorithm. The BTeV "vertex trigger" performs pattern recognition for every interaction in the detector (15 million /s)[7]. A WEB-based animation of the pattern recognition with explanatory text is available[8].

An overview of the architecture of the BTeV trigger and data acquisition system is shown in Figure 1, and some significant numbers that characterize the system are given in Table 1.

Table 1: Sizing characteristics of the system scale

# of Gbyte/s data links	Buffer memory	Data rate to L1 buffers	Data rate to L2/L3 buffers	Data rate to archive	# of L1 DSPs	# of L2/L3 Processors
2500	1 Tbyte	1.5 Tbytes/s	25 Gbytes/s	200 Mbytes/s	> 2500	2500

The trigger system has three distinct levels, called Level 1(L1), Level 2(L2), and Level 3(L3) and will use massively parallel computation pipelines at each level. L1 includes the vertex trigger, and additional triggers are being considered. Figure 1 shows the buffers that receive data from the BTeV detector, an expanded view of the first-level vertex trigger, a Global Level 1 trigger manager that will process the results from all first-level triggers, a switch that will route data to a large computing farm called the Level 2/3(L2/3) cluster, and the buffers and processors that make up the L2/3 trigger. There is no distinction between L2 and L3 hardware, since the same processors will be used to execute the L2 and L3 algorithms. A processor that receives data for an interaction will process the data using the L2 algorithm. If the data satisfy the L2 selection requirements, the data are processed using the L3 algorithm. Data that fail L2 or L3 selection requirements will be dropped. Data for an interaction that satisfies the selection requirements will be written to archival storage at an average rate of 200 Mbytes/s.

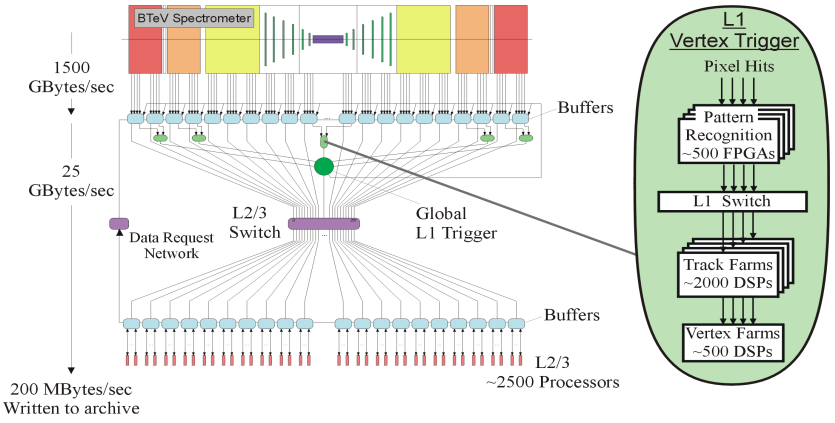

Figure 1: Schematic of the BTeV Trigger and Data Acquisition System showing (left side) the detector, buffer memories, L1, L2, and L3 clusters and their interconnects and (right side) a blowup of the L1 Vertex trigger

Detailed Monte Carlo simulations, data-flow simulations, and benchmarks for trigger algorithms have been performed to estimate the capabilities required of the BTeV trigger and data acquisition project. Here we describe the relevant results. As mentioned before, the data rate out of the BTeV detector and into the data acquisition system will be 1.5 Terabytes per second. All of the data for each interaction that occurs in the BTeV detector will be buffered long enough to give the L1 trigger time to decide if the data should be dropped or sent to an L2/3 processor. A subset of the data (the data from the silicon pixel vertex detector) will be sent to the L1 vertex trigger (inset shown in Figure 1). The L1 vertex trigger implements a pattern recognition algorithm using about 500 Field Programmable Gate Arrays (FPGAs) to find track segments in the pixel vertex detector (the first step in finding the particles that were created in an interaction). The track segments will be routed through a switch to a farm of Digital Signal Processors (DSPs) that reconstruct particle trajectories, followed by a second farm of DSPs that reconstruct vertices. Benchmarks of the reconstruction algorithms were performed for the Texas Instruments TMS320C67X DSP. The

results from these benchmarks and estimates of the input data rate indicate that the L1 vertex trigger will require about 2500 DSPs to achieve the necessary processing power. This estimate does not include additional processors for fault tolerance or for other L1 algorithms, nor does it include the large number of support processors required to configure, monitor, and control the DSPs. The design goal for L1 is to reject 99% of the interactions that occur in the detector. The interactions that survive L1 will be passed to the L2/3 trigger. L2 will perform a more refined analysis of the data and impose more stringent selection criteria than L1. L3 will improve the analysis for each interaction that survives L2 even further by considering data from each detector subsystem and by performing a detailed physics-based analysis to identify interesting interactions. We have estimated that the L2/3 trigger cluster will require on the order of 2500 general-purpose computers, such as Intel Pentiums running the LINUX operating system. The L2/3 system will provide another factor of 20 in rejecting uninteresting events for a total rejection of 2000.

The requirements for the BTeV trigger are well understood, and the design is at an advanced conceptual stage. However, there is still enough flexibility in the design to adapt to new discoveries and different implementation strategies. For example, calculations that are currently done in DSPs may be migrated into FPGAs or we may use more DSPs and fewer PCs , or vice versa. The supporting software infrastructure must be flexible enough to handle variations in the hardware design. There will be variations due to the availability of new and different types of hardware, the addition of redundant hardware for reliability, elimination of superfluous hardware, or algorithm changes that require different hardware implementations. We must retain the ability to make design changes that permit the most cost effective use of the computing hardware. **The results from the proposed research will provide critical feedback to the designers of the BTeV system to achieve the required robustness and agility.**

1.2 IT Aspects of the BTeV Trigger and Data Acquisition System

While the hardware platform required to achieve the above-stated goals is extensive and complex and the trigger algorithms are quite challenging, an even greater challenge is to keep the system functioning and producing quality results over a period of several years. The system must serve well during the detector commissioning and debugging stage, during routine operations, troubleshooting, and calibration. Even during normal operations, it must operate in several modes and switch between these modes dynamically. It must continue to operate in spite of the failure of some of its components. It must adapt itself to varying conditions in the Tevatron accelerator and in the BTeV detector. It must evolve as hardware is replaced or as more powerful components are introduced to extend its capabilities, and it must accommodate changes in software as more is learned about the physics and as the detector itself evolves. In addition to computational performance, the key requirements for the BTeV trigger system include: (1) Dynamic reconfiguration and partitioning; (2) High availability, including introspective, self diagnosing, fault tolerant, fault adaptive capabilities; and (3) Life-cycle maintainability and evolvability.

1.2.1 Dynamic Reconfiguration and Partitioning

The system must be able to handle two aspects of dynamic reconfiguration: the ability to dynamically adjust parameters of the experiment, and the ability to reconfigure and repartition hardware used to analyze and filter data from the experiment. The latter will require some degree of dynamic reconfiguration of the computation network, so that the system can vary both the interconnectivity between different trigger levels and the number of processors assigned to different tasks.

In order to execute a number of different tasks (some of them simultaneously) the system must be able to operate in different modes. Examples are: 1) standard trigger operation; 2) special modes to support commissioning, debugging, and calibration of the detector; 3) verification of repairs, upgrades, and replacement of hardware and software components; 4) in situ (re)calibrations as the experiment proceeds; 5) use of parts of the system to test new trigger algorithms; 6) verification of detector alignment and calibration at the start of each physics run, which can occur a few times per day; and 7) introduction of special diagnostic packages to investigate problems in the detector, the trigger hardware, or software.

There is another significant area where dynamic reconfiguration is highly desirable. Physicists will use offline computing resources, requiring extensive processing power and I/O capabilities, for data analysis. One possible platform will be the L2/3 trigger system. The L2/3 trigger is not always fully saturated by real-time data acquisition, since the Tevatron intensity decreases during a run cycle[9]. Therefore, the L2/3 processors should be available to physicists when they are not needed for trigger tasks. Research on cluster-based services for the Internet[10] has explored the use of *overflow pools* or sets of computers that can be used temporarily to handle prolonged bursts in demand for a service. We will explore a system that relinquishes resources during periods of low demand.

1.2.2 High Availability

The BTeV system must be available 24 hours/day, 7 days/week (24/7) to support all of the previously mentioned tasks. The highest possible data throughput and data quality must be maintained during normal trigger operations. Therefore, the system must be fault tolerant, introspective, self-diagnosing, and fault adaptive.

There are many ways that a complex heterogeneous system can malfunction, including the failure of individual processing elements or network switching elements. Moreover, a noisy accelerator environment or detector malfunctions could produce faulty data that impair the ability of the system to maintain data throughput or data quality. BTeV must be fault tolerant, but at an acceptable expense. For example, airline systems use two or three levels of redundancy in critical computer systems. This level of redundancy is costly, but the cost is warranted. In the case of the BTeV trigger, while it is unacceptable that the system loses large amounts of data, it is tolerable to lose a small fraction of the data for a short period of time. The key feature is that during fault conditions, the trigger system must continue to operate, possibly at decreased capacity (graceful degradation).

The system must be able to detect fault conditions, both locally and globally, and operator intervention should be kept to a minimum. Due to the system's complexity, it could take a very long time for an operator to recognize the existence of a problem, diagnose it, and remedy or mitigate it. All the while, valuable data from the experiment would be lost, or be of poor quality. The system must take the initiative to mitigate and adapt to faults.

One basic operation when adapting to faults is to remove the failed component from its partition, and restart its computation on a similar component. An example of this is a classic offline and batch processing system that relies on check-pointing. We will evaluate at which levels check-pointing is appropriate for high performance embedded systems, where loss of data, rather than compute time, may occur.

The proposed BTeV system has the capability to change many key operating parameters to repair or mitigate problems detected by its analysis programs, not only within the trigger and data acquisition system but also in the detector complex. For example, it can reduce the high voltage supplied to a noisy detector, or raise the threshold for deciding that it has valid information. Furthermore, there are typically several physics triggers and several calibration and monitoring triggers, which are "pre-scaled" to obtain an output data stream. The system can adjust these pre-scaled triggers or turn off lower priority triggers to preserve resources - computational, memory, or network bandwidth - for the highest priority data.

1.2.3 Life-cycle Maintainability and Evolvability

BTeV will be constructed over the next 5 years and will then run for at least 5 years after that. It will be essential to develop a trigger and data acquisition system that can be easily operated and maintained. It must be a system that can evolve as old hardware fails or becomes unsupported and as new hardware and software technologies that offer improved performance become available. In addition, the experiment itself will undoubtedly be modified to respond to new ideas and challenges as we learn more physics. The system must be able to adapt to support these changes. One has only to look back over the experience in HEP in the last decade to understand how difficult it will be to achieve this goal, and how necessary it is to address it from the earliest design.

2. Project Description, Goals, and Objectives

The BTeV trigger system is an example of a massively parallel, high performance, high reliability, computational system. The design and implementation of such systems cannot be achieved by the ad hoc approach of developing simple small-scale components and scaling them up into large-scale systems[11]. Issues such as fault tolerance and performance must be explicitly addressed at multiple levels in the system design[12]. We propose advances in system design methodology, tools and runtime infrastructure to facilitate these and more issues involved in developing such systems. We further propose to develop the software to accomplish the design and implementation of the system and to study its performance, utility, and scalability on the actual BTeV hardware as it grows over the construction phase of the experiment. **The result of this research will be software, design methodologies, and the documented experience of the project**.

Several capabilities are required: (1) **System Modeling and Analysis –** Full-system performance estimations are needed during development, given the coupling that exists between different aspects of a system design (e.g. low-level architectural decisions can have a large impact on system-level performance and fault behavior). Designers need mechanisms for representing and evaluating the impact of these decisions. The design tool will serve as a framework for modeling and analyzing system designs via behavioral simulation, performance simulation, design verification, etc. The design tool will continue to be useful during operations to understand how to handle unanticipated situations, which often arise in HEP research; (2) **System Configuration Management –** Configuration of a large-scale networked processing system is a complex problem, more so when the system is susceptible to faults[13].

A robust configuration management infrastructure is required, with the ability to specify reconfiguration strategies at different levels. The fault mitigation infrastructure is intricately coupled to the configuration management infrastructure – means to capture the specifics of the coupling are necessary; (3) **Runtime Environment and Hierarchical Fault Detection/Management –** The deployment, execution and reconfiguration of the components must be carefully managed, especially when the cost of downtime is high, as is the case for BTeV. Runtime environment control is essential. A system-wide infrastructure is required for rapidly detecting, isolating, filtering, and reporting faults. In very large-scale heterogeneous systems a single centralized fault management solution is clearly not feasible[14]. Hierarchical distributed fault mitigation is necessary, with the ability to specify fault mitigation policies at different levels of abstraction (system, network, node, etc.)

The architecture of the framework is shown in figure 2 and is discussed below. There are two principle components of the proposed framework: In Section 2.1, we will introduce the Design and Analysis Environment, and in Section 2.2 we will describe the Runtime Environment. The proposed framework leverages work both in design automation tools, and in real-time, parallel, and fault tolerant runtime systems to support system design, analysis, and implementation.

2.1 Design and Analysis Environment

A high-level design tool is required to support the overall design, deployment, and evolution of BTeV-type systems. The Model Integrated Computing (MIC)[15,16] approach, developed at the Institute of Software Integrated Systems (ISIS), Vanderbilt University, assists the creation of domain-specific modeling, analysis, and program synthesis environments for building complex, large-scale computer-based systems. Integrated models created in this environment represent all relevant factors of a physical system. Models can be subjected to many types of rigorous analysis, for verifying the behavior and performance of the system prior to implementation. Central to this approach is the concept of a "configuration", which is a particular organization of computing resources, such as processors, network components, memory buffers, and storage elements, and a particular allocation of software components and datasets on them, including task schedules and message routes. Systems can be synthesized (or generated) from the models when the designer is satisfied with the analysis results. The Design and Analysis Environment will consist of: a) graphical modeling language/environment for system specification; b) synthesis tools for interfacing the models with commercially available and custom analysis tools; and c) synthesis tools for generating configurations from specifications, for configuring fault managers, and for configuring operation managers. The Design and Analysis Environment is the part of the system shown above the dotted line in Fig. 2.

The synthesized configurations are deployed and executed in the Runtime Environment. The primary interaction between the Design and Runtime environments is through the synthesis process. However, feedback from the Runtime to the Design environment is possible in advanced fault scenarios that require re-synthesis and redeployment. ***While MIC provides the basic infrastructure, research is required to define: (1) modeling language and composition methodologies suitable to BTeV's application; (2) mapping techniques for models to/from analysis tools; (3) large-scale synthesis techniques.***

2.1.1 Modeling

We will develop a multiple-view, domain-specific graphical modeling paradigm to allow physicists to specify BTeV systems. We will compose this paradigm from best-of-class modeling formalisms, and models of computation[17], the composition ***yielding a unique integrated formalism***. Following are the main modeling views and their associated formalisms:

Information/Algorithm Data Flow Modeling

The **Algorithm Modeling** view describes the experiment's processing algorithm structure. The algorithm is modeled as a **Dataflow Graph** with the nodes representing experiment computations and arcs for communications/data routing. **Hierarchical Composition** manages complexity, and allows system partitions to be designed independently, and assembled on an experiment-specific basis. **Design Alternatives** express implementation options, which have been previously used in research for system optimization and will be extended to support fault mitigation strategies. Timing information will be captured (see **constraints**). Timed Data-flow representations exist for real-time systems[18], however no provision for fault behavior was included. The dataflow graph will be extended with **Data Routing/Distribution Modeling** to manage complex, data-dependent routing (necessary for extreme bandwidth systems), addressing the real-time distribution of information within the system. **Explicit Control Flow** modeling will permit specification of control actions for experiment data processing. The **Software Failure Propagation** view will define expected exceptions, their severity/priority, and their propagation (e.g.*, Div By Zero, Real-Time Deadline Miss, Pointer-out-of-range/bus error, User-defined/data dependent exception*.) Failure propagation

graphs will be superimposed upon the computational structure to capture the impact of software component failure on other software or hardware components. Failure propagation graphs have been applied to physical system diagnostics[19,20]. ***Their application to software failures is a new research approach***.

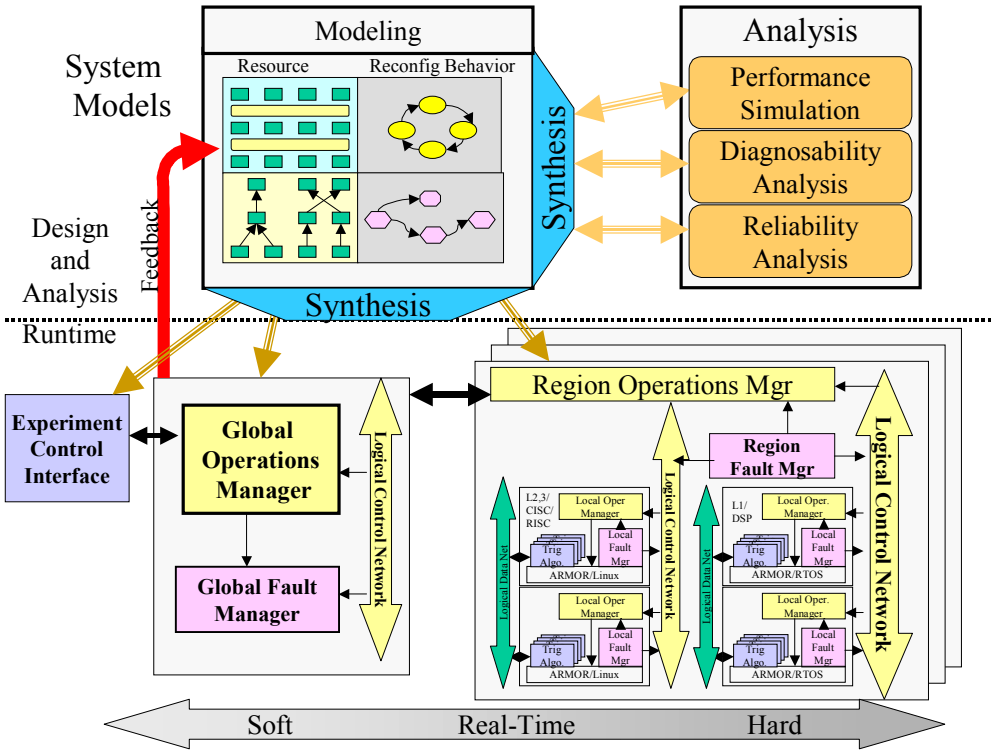

Figure 2: **Bi-Level System Design and Runtime Framework – System Models use domain-specific, multi-view representation formalisms to define system behavior, function, performance, fault interactions, and target hardware. Analysis tools evaluate predicted performance to guide designers prior to system implementation. Synthesis tools generate system configurations directly from the models. A fault-detecting, failure-mitigating runtime environment executes these configurations in a real-time, high performance, distributed, heterogeneous target platform, with built-in, model-configured fault mitigation. Local, regional, and global perspectives are indicated. On-line cooperation between runtime and modeling/synthesis environment permits global system reconfiguration in extreme-failure conditions.**

Target Hardware/Resource Modeling

Resource Modeling allows us to define the hardware platform. It is modeled in terms of: **Node Architecture**, capturing the attributes of a computational (DSP, CPU, Memory) or routing (FPGA) node; **Topology,** the connectivity, bandwidth, timing, and messaging protocols; and **Routing Infrastructure**, the capabilities of data merge/distribute components. For data-intensive systems, *active routing* is required, where data distribution/management is algorithm-specific (i.e. FPGA processing elements in the data path). Capturing this information explicitly in models will facilitate performance optimization. **Hardware Failure Propagation** models capture the anticipated failure modes of the processors and switches, along with their severity and propagation characteristics. These modeling views combine to form the 'Target Hardware Configuration' of the original, non-faulted computer system. The resource models will be dynamically updated by the Runtime Environment in advanced fault scenarios. ***Node architecture and topology is based on prior work; failure propagation is new in this context***.

System Detection and Fault Mitigation Modeling

Fault mitigation models capture potential fault scenarios and user-defined fault response strategies as **Hierarchical Finite State Machines (HFSM)**[21] to help resolve and mitigate faults. The HFSM states represent responses of the BTeV system executing management actions, such as: reschedule a process on another processor; reroute data around a faulty communication device; disable a sensor input; re-synthesize from an existing system, performing

minor perturbations; or globally re-synthesize an optimized system (i.e. when the fault is not recoverable with a simple action).

Rescheduling, rerouting, and disabling actions can rapidly (milliseconds) handle minor component failures, with an anticipated degradation of the system capabilities. Perturbation re-synthesis will locally re-optimize for failures and attempt to achieve minimal system degradation, executing in tens of seconds. Global re-synthesis will make best use of all available hardware, aggressively rescheduling/reallocating to maximize functionality in the event of significant component failure, with longer recovery times (minutes-hours).

System Constraint Modeling

Many requirements for the system design crosscut modeling aspects. These requirements are expressed as **Constraints**. Constraints play two important roles: a) they express relations, complex interactions, interdependencies between objects and properties in different aspects, and b) they express non-functional requirements. An extended variant of the Object Constraint Language (OCL)[22], has been developed[23] to express the following forms of constraints: (a) performance constraints, (b) resource constraints, and (c) compositional constraints. ***The application of a constraint language to system partitioning and allowable fault-mitigated system configurations is new research***.

2.1.2 Analysis

Prior to system synthesis it is important to subject the designed system to a variety of rigorous analyses to help uncover subtle design flaws. When diagnosed at a late stage these flaws lead to costly redesign. In previous MIC research, the design framework was interfaced with a variety of analysis tools for diagnosability (DTOOL for the International Space Station)[24], performance simulation (PML discrete event simulation for Adaptive Computing Systems Model Integrated Design Environment)[25] and state space analysis for safety and reliability (SSAT for Sandia National Labs, analysis of High Consequence/High Assurance Systems)[26]. We will extend these efforts to address the properties and requirements of the BTeV system. In particular we will apply these analysis techniques to timing, performance, bandwidth requirements, and fault tolerance. The results of the analysis will be fed back into the system models[27].

2.1.3 Synthesis

Synthesis involves several phases: 1) Resolution of design alternatives, partitioning and processor allocation; 2) Generation of system configuration source code (communication maps, processor schedules, glue code, programmable hardware designs (VHDL/Verilog), etc.); 3) Compilation/linking to executable modules/hardware bit files. Loading/deployment is done by the Runtime Environment.

Design Alternative Resolution, Partitioning, and Processor Allocation

The system models capture a multi-dimensional design space. From this space, a few configurations must be selected which satisfy the design objectives while not violating any constraints. **Design Space Exploration** has been used in the past to select configurations for performance optimization[23]. In BTeV, the configuration space can be explored for fault mitigation (i.e. selection of configurations that do not rely on faulted components.) ***The size of the design space, and complexity of constraints makes this a challenging research issue.***

We will study, adapt, and extend at least three approaches for design alternative resolution, partitioning, and processor allocation. (1) Ordered Binary Decision Diagrams (OBDDs)[28,29] for constraint satisfaction; (2) Multi-agent-based processor allocation/partitioning[30]; and (3) Economically rational decision-theoretic model for processor allocation and scheduling[31].

System Configuration Generation

The next phase of system synthesis generates configuration artifacts to be used in creating system executables. These include compilable design specifications for configurable hardware components (FPGAs); schedules, communication maps, message routing tables for commodity/DSP processors; specifications for operation and fault managers. These artifacts are compiled to generate system executables. Specifically:

Hardware*:* For some configurable hardware components within the system, specifically programmable routing elements (FPGAs), design specifications will be generated. The design will be composed of existing components (from a standard interface runtime library or hand-developed/commercial IP) using structural VHDL/Verilog and a set of glue 'intrinsics'. Off-the-shelf tools generate "bitfiles", specifying the electrical structure of all configurable chips. *We will leverage the ACS program results in hardware synthesis*[32].

Software*:* For software-programmable (L1 DSP and L2/L3 general-purpose CPU) components, software specifications are generated, providing information for the Runtime Environment to implement the necessary computational behavior. Specifications include: real-time schedules and communication maps for information flow. Interfaces be-

tween software modules and hardware modules/data sources/sinks are automatically inserted, managing the hardware interfaces for complex communication protocols into simpler hardware compatible protocols, multiplexing physical ports and performing data conversion functions. *We will leverage the ACS program results in software synthesis*[33].

Operation & Fault Manager*:* The fault detection and mitigation behavior models are used to automatically synthesize/configure a hierarchy of operation and fault managers. Hierarchical fault managers are configured with the specified behavioral descriptions. State tables and next state equations and the interfaces to internal and external events are generated, for the runtime executables. The fault managers interface with operation managers at each level in the hierarchy, performing local, regional, or global reconfiguration.

Partial/Online Re-synthesis: In addition to a full-scale optimized system synthesis, the framework will also support small-scale, rapid perturbations. Small hardware failures trigger limited-scope system re-synthesis. A cost function associated with system modifications will be assessed to balance the benefit of a modification to data quality vs. the cost to reprogram system components. ***On-line partial re-synthesis is a research issue.***

2.2 Runtime Environment

2.2.1 Operating System(s)

The runtime environment must support real-time scheduling, message passing, synchronization, and some preliminary fault-tolerance mechanisms for the DSP elements of the Level 1 trigger. The L2/3 computers must have support for "soft" real-time response, since although they are part of the data path, there will be less strict timing constraints at these levels, and more flexibility in the scheduling of tasks. To best use existing work, we propose to extend available OSs to meet the needs of the BTeV system. The extensions will be done above the OS whenever possible, but kernel modifications will be employed when dictated by the latency requirements of the system.

There are several real-time operating systems (RTOSs) that we can consider for DSPs: Texas Instruments' DSP/BIOS[34]; ENEA OSE real-time kernel[35]; and some non-commercial university kernels[36]. We will investigate which of these can be used in BTeV at different levels. A primary requirement is that the chosen RTOS be able to accommodate the application and constraints of BTeV. We intend to use our expertise to extend the chosen RTOS with fault tolerance, reconfiguration, and partitioning capabilities. As shown in Figure 2, the DSPs at the Level 1 trigger will use DSP-friendly embedded hard RTOSs. The trigger Levels 2/3 do not have hard real-time requirements and Linux (or Linux/RT) is envisioned as the primary operational platform.

2.2.2 Runtime Hierarchy

As an extension to the operating system, and following the hierarchical structure of the rest of the system, we introduce a hierarchical computation and communication infrastructure responsible for (1) runtime resource monitoring, (2) dynamic load-balancing of the entire system, and (3) distributed hierarchical error detection, identification (diagnosis) of failed components, and recovery (including reconfiguration) from errors that affect the hardware, the operating system, or applications.

The environment considered in this work is highly dynamic and operates under continued stress in terms of faults and varying workload (data traffic and computation intensity). The challenge is to provide system solutions (algorithms and architectures) that can guarantee delivering a required level of service in spite of errors and the unpredictable dynamics of the system. Detection and recovery will take time and an appropriate level of response must be provided, corresponding to the reliability and temporal requirements of each application at each level.

Moreover, the infrastructure will monitor the performance of computation processes throughout the system (migrating them when necessary), provide (at least) soft-real time guarantees, and handle its own errors (i.e., the infrastructure must be self-checking). Finally, the proposed infrastructure should dynamically adapt to the changing operational conditions (e.g., fault/error rate and variability in workload) and should enable seamless transition between full operational capabilities and operation in a degraded mode. It is essential to keep the application alive until the full computation capacity of the system is restored. Due to the hardware and operations cost of BTeV, graceful degradation is a very cost-effective solution for high availability. This infrastructure will interact with the global operations manager and the global fault manager as depicted in Figure 2.

2.2.2.1 Very Lightweight Agents (VLAs)

At the lowest level of the runtime hierarchy, we will apply the concept of *very lightweight agents* (VLAs). Applied to DSPs, these are simple software entities, which can be implemented in a few dozen lines of assembly language, that take advantage of the exception-signaling and interrupt-handling mechanisms present in most DSP kernels to expose errors in the kernel behavior. When the VLA detects (e.g., by monitoring DSP exception signals) an error condition, it reports to an ARMOR (described next), which takes appropriate actions such as disabling the exe-

cution thread or discarding the current data item. A similar mechanism will be explored for the monitoring and reporting of deadlines, traffic, processor loads, etc. Moreover, the interrupt mechanism will also be used to trigger reconfiguration of the software or hardware at this lowest level of the hierarchy. Note that since the software VLAs are small and interrupt-driven, the latency introduced by VLAs will be negligible.

Hardware VLAs can also be developed for FPGAs, consuming only small number of gates, and taking advantage of otherwise present communication resources. ***VLAs (software and hardware) in this context is new research area.***

2.2.2.2 Adaptive, Reconfigurable, and Mobile Objects for Reliability (ARMORs)

The fault tolerance and performance-oriented services offered to the system will be encapsulated in intelligent active entities (agents) called ARMORs (Adaptive, Reconfigurable, and Mobile Objects for Reliability). ARMORs are, by design, highly flexible processes, which can be customized to meet the runtime needs of the system[37]. Variants of ARMORs will run on DSPs, L2/L3 processors, and other supporting processors throughout the system.

ARMORs communicate through message passing and all functions of an ARMOR process and its runtime behavior are encapsulated in *elements* (see Figure 3(a)). Elements constitute basic building blocks, which usually encapsulate elementary detection and recovery services available to the application. New functionality can be introduced into the system without disturbing existing functionality, as long as, from a resource or timing perspective, it does not affect the current system. In other words, the resource manager must implement some type of resource protection. Services provided by the elements are invoked by the ARMOR interface, which serves as a communication gateway with the outside world. The ARMOR interface has two primary responsibilities: (1) controlling the addition, removal, and replacement of constituent elements within the ARMOR, and (2) providing communication among ARMORs.

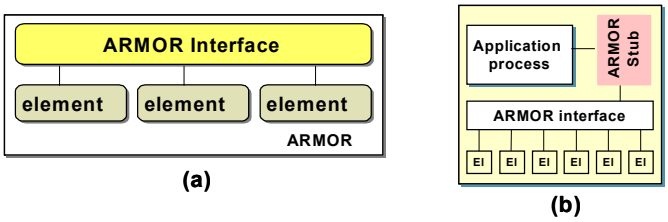

Figure 3: (a) ARMOR architecture,(b) Embedded ARMOR

An application can take advantage of ARMOR provided services (such as error detection and recovery) through the concept of an *embedded ARMOR* in which the core element structure of the ARMOR is linked to the user application process (see Figure 3). The application code is lightly instrumented with the embedded ARMOR API to invoke the services provided by the underlying elements. In this configuration, the embedded ARMOR process appears as a full-fledged ARMOR to other ARMORs in the system and as a native application process to non-ARMOR processes. **This permits BTeV physics applications to use the same apparatus for error handling as will be used to handle errors within the computing platform itself.**

2.2.2.3 Hierarchical Detection and Recovery

Designing a hierarchy of detection and recovery requires a balance between placing disparate techniques in separate layers and minimizing the overhead due to the interactions between multiple layers. Also, the detection and recovery layers must be evaluated with respect to their latency, the performance degradation incurred, and their coverage. This would guide the decision as to which of the detection and recovery mechanisms are best suited for specific classes of applications.

There is a strict synergy between this hierarchy of error detection and recovery in the hardware structure of BTeV, with the hierarchical software structure that we are developing (best seen in the fault managers and the operations managers in Figure 2). This synergy enables the fault-tolerant subsystem to be incorporated in the system as an integral part, not as a cumbersome afterthought.

An ARMOR with embedded techniques for detecting and recovering from errors serves as a defense line against system failures. The detection techniques are arranged in a hierarchy of logical layers[38]. The chain of detection starts at the lowest layer, and violations not captured at a lower layer bubble up to higher layers. The lowest detection layer includes techniques internal to the ARMOR (e.g., signature checking and livelock detection) and makes the ARMOR substantially self-checking. The next layer provides error containment within a node and makes the node fail-silent to the outside world (e.g., exception handling, smart heartbeats). The next layer detects failures among multiple ARMORs and ensures that all processes have a consistent view of the outside world. In an example recovery scenario, after detecting an error, an ARMOR would create a corresponding error context, which is forwarded to the error handler. The error handler interprets the information in the error context and initiates recovery action(s) according to predefined rules and strategies.

In designing hierarchical error detection and recovery it is essential to ensure adaptability of individual layers. The detection and recovery invocation conditions at the individual layers should be customizable to application needs, the types of faults being experienced in the system, and the reliability characteristics desired for the system. When designing recovery strategies, it is important that one and only one ARMOR be responsible for the ultimate recovery of a failed entity in order to avoid contention during error recovery. Independent of which layer of the hierarchy handles an error, the information about the occurrence of the error is propagated upwards through the hierarchy so the global fault manager has a consistent view of the system behavior.

Figure 4 illustrates error detection mechanisms encapsulated in three logical layers. Because each layer can incorporate a suite of detection techniques, a layer is characterized by the specific techniques and also by the location in which the techniques execute. In this research we will define a hybrid (location and application-defined) set of techniques that can be used within the resource and timing constraints.

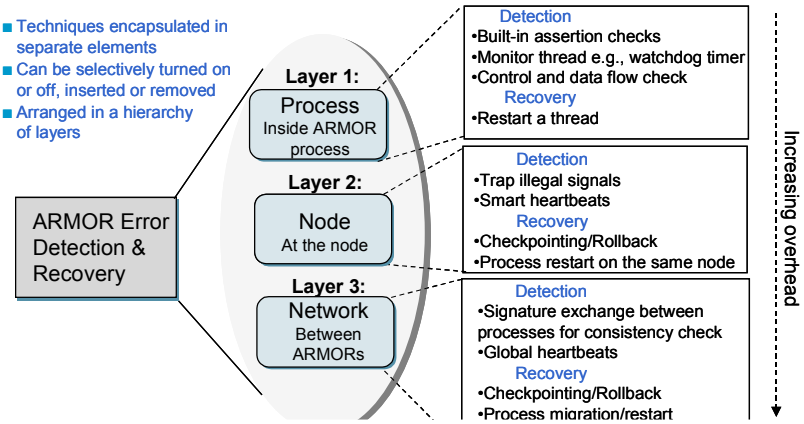

Figure 4: ARMOR Error Detection and Recovery Hierarchy

As in the case of error detection, it is possible to consider logical layers for recovery as well (see Figure 4). We define three logical layers of recovery: (1) within a process, (2) within a node by the monitoring process on the node on which the application resides, and (3) a global recovery initiated by a global fault manager. In the context of these three layers of recovery it is critical to ensure that one and only one recovery path is active at any time for a detected error.

Note that the time consumed by resources in carrying out error recovery has to be accounted for by the operations managers. Information regarding detection and local recovery must percolate up toward the higher layers of the hierarchy, so that more accurate diagnostics and recovery can be attempted. These "more intelligent" and more time-consuming tasks can be either accounted for at design time, or execute during slack in the system. Either way, the distribution of new reconfiguration or recovery procedure information will use shared resources and therefore has to be taken into account. Lastly, the high-layer detection and recovery may take the resource usage into account when developing new strategies for dealing with the faults and with reconfiguration for graceful degradation.

2.2.3 System Validation

Evaluation of the proposed infrastructure must be considered as an integral part of this work. An efficient and accurate evaluation of attributes, including reliability, availability, and performance, requires a comprehensive and well-established methodology. To address this challenge we need to understand, in collaboration with BTeV physicists, the complex nature and variety of unexpected conditions that can affect the system, and to explore the interactions between system fault tolerance and performance and their combined impact on the service delivered by the system. This knowledge is essential to produce realistic system stress that will activate and exercise mechanisms dedicated to cope with system anomalies and will provide insight into the system bottlenecks.

In order to fully stress the system we will provide: (1) an approach to generating stresses (e.g., faults) that are realistic representations of operational environments, (2) a common mechanism to inject the generated stress, and (3) stress models and injection strategies that are scalable to new hardware and software technologies, and system expansions. To conduct evaluation studies, we will use software-based techniques for fault/error injection for creating failure scenarios representative of real operational conditions. To this end we will leverage our experience in developing NFTAPE[39], an environment for conducting automated fault injection experiments. Considering the complexity and scale of the system discussed in this proposal we need to significantly enhance fault injection capabilities by adding support for new fault/error models.

2.2.4 Collection of Data for Creating and Validating New Fault Models

A less-glorified fault tolerance related issue is the collection of fault/error data for *a posteriori* analysis. We will develop new, low-overhead techniques to do data logging and aggregation of failure data, with the help of the

VLAs and the ARMORs. This will provide important feedback to designers of both software and hardware, and guide future iterations of BTeV. By collecting such data we will be able to derive the failure rates for each type of element used in BTeV, and to correlate the faults/errors among different components. We intend to use this data to produce realistic fault models for use by the fault tolerance community at large. Furthermore, we will investigate the creation of fault tolerance benchmarks that could be distributed via the World Wide Web. The new fault models will also enhance capabilities of tools such as the NFTAPE[39] fault injection environment developed by the UIUC partners, and the FT-RT-Mach[40] operating system developed by the University of Pittsburgh partners.

2.3 Other Aspects of the Project

The system described here is only the core of the fault-tolerant, fault-adaptive framework that supports the BTeV trigger system. There will be many supporting hardware subsystems that will control DSPs and manage hardware settings throughout the computing and detector systems. The configuration management and fault tolerance of the trigger farms and supervisory system must interface to the experiment specific code on several fronts:

Run management*[41]*: Some successive runs, such as normal data taking, may have identical configurations that simply require triggers to be re-enabled, while others, such as special calibration runs, may require large reconfiguration of control constants and a different suite of application programs.

Persistent storage (resource management, run history): The run configuration history (which hardware participated, versions of trigger algorithms used, etc.) must be stored into the run history database. The current hardware configuration must be accessible to experiment specific software.

User interface/diagnostics*:* Runs are started, stopped, and monitored by human operators. The experiment needs access to the configuration, status, and error information to interface to the control room GUIs. Diagnostics and status information at each level of the system must be provided to trace and diagnose the problems.

Application code: The application code will have status and error information, which should use the same underlying infrastructure that is built into the fault-tolerant system. The ARMOR application interface provides this capability. These issues all have to be addressed in the wider context of BTeV's online and runtime system but are too detailed to discuss here.

3. Relation to Existing Research

The main thrust of the research proposed here is to develop runtime software and design methodologies (as well as the document our experiences with BTeV) for large scale and complex real-time, embedded, fault-tolerant systems. The target of previous research for embedded systems are far from the magnitude of BTeV and do not present the software challenges or operational real-time reliability issues of this degree. Several efforts[15,33] have laid the foundation for the development of this framework, but have not addressed the mapping techniques for such heterogeneous and large-scale systems (100s of FPGAs, 1000s of DSPs, 1000s of L2/L3 computers, different types of network hardware, global managers, etc), the hierarchy of fault management, or the information percolation techniques suggested here.

Other works have addressed load balancing for performance purposes[42] but mostly for distributed systems with loosely coupled elements and much higher latency requirements than the 132 ns timeframe we are faced with in BTeV. The mapping of tasks to processors has been treated in several works previously, based on constraint languages[23,43,44,45], constraint satisfaction[31], and distributed constraint satisfaction approaches that are market based[46,47], evolutionary algorithm-based[48,49] and decision tree-based[29]. Some works have attempted multi-agent scheduling[30], but the agents that satisfy the required aggregated behavior are not scalable.

From the software perspective, we will expand on the state-of-the-art in software synthesis[50] and apply the new findings to large-scale systems, which will encompass the code generation for the items discussed above (partitioning, mapping, fault tolerance, etc). These synthesis tools have been applied to physical systems[32], but not to software failures and failure propagation so far.

From the runtime perspective, many DSP-friendly kernels come with scalable extensions that support real-time scheduling and synchronization, host-to-target communication, or real-time instrumentation. At the higher levels (L2/3), real-time kernels are common in the industry (QNX[51], VxWorks[52], Linux/RT) but fault-tolerance has only been addressed in this context for transient conditions. We will choose the kernels and runtime environments that best fit the different types of subsystems being addressed, and create the necessary software tools to compose them into a unique framework.

Many of the existing solutions for providing fault tolerance in a network of unreliable components are based on exploiting groups of replicated processes, e.g., ISIS[53], Horus[54], and Totem[55] where the fault tolerance is something of a side effect of the replication approach. Examples of systems which explicitly address the issue of fault tolerance

include Delta-4[56], Piranha[57], and AQUA[58]. Most existing systems require a complex software layer and/or additional hardware to provide group communication and good coverage for fail-silent behavior of processes and nodes. Several of them detect failures solely through the use of timeouts.

The hierarchical scheme proposed here creates a foundation for providing efficient mechanisms to monitor performance, to detect and recover errors, and to install configurations for new experiments and for fault mitigation. The introduction of VLAs and ARMORs and their system-wide use offers configurable degrees of fault tolerance for both applications and the infrastructure itself, and is a significant step in merging the research in the areas of distributed systems and distributed agents. The hierarchy of VLAs and ARMORs not only enhances error detection and recovery of the overall system when compared to schemes used by existing solutions (mentioned above) but also enables collection of the error/failure data. This allows the construction of new fault models, which are currently nonexistent. We will leverage our positive experience in employing the ARMOR technology in the JPL REE project[37,59,60]. The scheme proposed here is of much larger scale. The challenge will be to find and assess methods/techniques that can handle errors under stringent temporal and spatial constraints.

4. Applicability to Others Areas of Computing

The BTeV trigger system is a large scale, real-time, embedded computer system. In order to achieve the high throughput, the large rejection of unwanted events, and the high efficiency for a small subset of rare and crucially important events, many of the usual High Energy Physics (HEP) specific features normally associated with such systems had to be eliminated from the design. For example, all sensor data is stored in a large (>1 Tbyte) buffer memory. Once in the memory, the data are seen by the system as an abstract data source whose detailed nature is significant only to the processing algorithms. To achieve high throughput, there is no fixed latency anywhere in the system and there is no requirement for events to emerge from the system in the same time sequence as they occur. These features give the system a generality that extends well beyond HEP. Previous information technology work has explored fault tolerance but the scale of this system poses new problems and requires the evaluation of new technologies and new design approaches. These issues are applicable to a wide range of scientific and engineering computing applications. Here we briefly discuss three classes of applications that will be able to take advantage of the work done in this project.

The first is a class of smaller scale but nevertheless ambitious projects in astrophysics. A typical example is the Pierre Auger project[61].

The second class is in the area of medicine: PET (positron emission tomography) scanning is a technique of introducing radio-nuclides into the body and following their progress by detecting the photons they emit. The radiation doses involved in this process are significant. PET field-of-view and data acquisition systems are very inefficient by HEP standards and could be improved. The data rates and real-time control problems would benefit from the applications of the methods developed in this project. The goal is to improve the efficiency of the system and extend the devices to larger field of view (more data) so that this procedure can become a basic diagnostic tool rather than a tool employed only to investigate specific problems uncovered by other methods.

Systems to image neural activity are now being undertaken on an unprecedented scale. The hardware to do the imaging is well advanced but the data acquisition systems needed to keep up with the ever increasing data-flow are simply not available. Moreover, the need to employ rare and short-lived tissue places strong demand on the systems for rapid feedback for real-time control of the experiments as well as fault-tolerance. Researchers in this field should be able to adapt the software and experience of this project to their needs[62].

Other applications that could benefit from high computational performance and fault tolerance are: monitoring turbine engines and rocket motors, global weather monitoring and disaster early warning systems, satellite based surveillance, autonomous vehicle navigation, computer vision, highly available Internet based services, and air traffic control systems. To explore the relevance of this work to other application domains, we will hold a series of 4 workshops of 2-3 days duration. For the first workshop, about 6 months into the project, we will invite a dozen experts from selected applications areas that might be able to use this work. This should result in interest in the project from one or more participants. We will follow this up in the second and third years with workshops to update and receive feedback from users. At the project's end, we will hold a workshop to explain and summarize what we did to a broad set of prospective users. Funding (travel, lodging) of invited participants is in the budget. Collaborating institutions will host the workshops and provide the remainder of the support.

5. Educational Component

As part of this proposal, an active involvement of high school teachers will be implemented, modeled on the existing summer teacher research programs at Fermilab and the participating universities. Starting with the theme

"Why don't things always work as well as we'd like", the teachers will design web-based displays that will allow students to learn such basic concepts as exception handling and fault tolerance and how they can improve the quality of products and services. The simplest exercises might, for example, involve designing a "scheduler" for one of a set of common activities, such as athletic activities, drama performances, etc. Students will be encouraged to discuss what can go wrong and methods for dealing with them. Common fault mitigation techniques such as having an understudy in a theatrical production will be identified. Students with computer programming skills will be provided guidance to write scheduling programs that can be confronted with various real world problems to see if they are prepared to handle them. We will teach the students error detection and recovery techniques and discuss how difficult it is to predict and address all the situations that can arise. The most advanced students can use actual BTeV simulations to "debug" system faults, just as experimenters will do online. The goal is to teach students at all levels of technical sophistication that making a system that works involves thinking carefully about how it can fail and how the failures can either be avoided or dealt with. The materials will be pilot projects, based on educational standards, which allow apprentice experience with real data and have the widest possible distribution. **A special feature of this proposal is to make internet technology relevant experiences and learning available to the predominantly physical sciences-based QuarkNet[63] program**, which already involves 275 teachers nationally (adding another 144 next year) in developing web-based instructional materials. We will also exploit connections of our university groups to programs in their area[64].

At the university level, training of undergraduates and graduate students will play an important role in supplying industry with people well prepared to advance the technology in designing and implementing highly available, high-performance computing systems. We will leverage experience from this work to create model courses and enhance existing classes on design of high availability network systems. This will create an opportunity to disseminate knowledge about and raise public awareness and understanding of reliability issues. We will seek an involvement from undergraduate students interested not only in computer science but also in applying information technology in experimental work. In particular we envision undergraduate projects, which would focus on: (1) investigating techniques for error detection, isolation, and recovery in high performance, network systems, (2) analysis of tradeoffs between the level of availability/reliability and performance overhead due to error handling mechanisms, (3) exploring methods for characterizing failure behavior of complex networked systems, (4) studying relations between a system configuration and an experiment setup the system is supposed to support. In addition to special programs, existing courses will be adapted to expose students to these topics[65].

6. Project Organization, Milestones, and Deliverables

The project will consist of a team of computer scientists and software engineers, largely university based, along with application domain specialists (BTeV-associated physicists and software engineers). The Principal Investigator (Paul Sheldon of Vanderbilt) for this proposal will be responsible for the overall project. He will have a steering committee consisting of the chief software engineer for the project, a scientific representative of BTeV, the leader of the BTeV trigger and data acquisition project, and the project education liaison. The chief software engineer will chair a technical committee. The project will be carried out as a coordinated but semi-independent set of research topics tractable for small teams at universities. Tasks will be assigned based on recognized milestones and deliverables and resources will be allocated to the participating institutions based on memoranda of understanding. There will be regular (at least quarterly) collaboration meetings and regular status reports. The PI will set up an external review committee to ensure the work is going well and has general application outside BTeV.

We will take advantage of the Fermilab's investment in R&D for the experiment to carry out large-scale testing on prototype hardware. We also will be able to employ the software on the full system that will eventually be funded through the experiment. Hardware purchases for the project will be limited to, at most, the provision of relatively modest test systems located at the participating university groups.

The preliminary and non-exclusive breakdown of responsibilities among collaborating institutions is: Modeling and Design framework: Vanderbilt with input from Syracuse and Pittsburgh in the partitioning, load balancing and task allocation parts; Runtime Fault Tolerant System: Illinois, Pittsburgh, and Syracuse, combining the VLAs and ARMORs to create the system hierarchy; Interface to BTeV and run control and monitoring: Fermilab; Trigger algorithms, physics applications, and input on operating conditions: BTeV physicists and software engineers.

The main deliverables of the project are: (1) the design and modeling software and the runtime system software; (2) the results of all intermediate investigations and studies of interest to people outside the project, as captured in reports, articles, and logs; (3) benchmark performance results from the test-bed systems that are of interest outside the project; (4) Web displays, course materials, reports and captured experience which form the educational component; and (5) reports and results from at least two applications outside BTeV of the system along with the records

and results of the applications workshops. The high level milestones of the project are given in Table 2. The results of this work will be made available to the public through publications in journals, articles, and lectures. Public Web pages will present many aspects of the work. Educational materials will be available from the Fermilab Education Office and the QuarkNet program and through university groups. Software and documentation will be available from the FermiTools public distribution[66] and the university groups.

Table 2: Preliminary Project Milestones

Funding Year	Design Environment Milestone	Run time System Milestone
FY1Q1/2	Modeling language and env (preliminary) Specify Interface to Run time env.	Design of overall runtime system hierarchy (ARMOR + VLAs)
FY1Q3/4	Synthesis of operations and Fault managers DSP and LINUX Synthesis	Design and implementation of VLA & ARMOR prototypes
FY2Q1/2	Modeling language and env Design space (preliminary)	Communication structure between VLAs and the levels above
FY2Q3/4	Synthesis of performance simulator Synthesis of all operations mgrs (final) Hardware synthesis	Detection and recovery in Layer 1 of ARMOR. Study Dynamic load-balancing (DL)
FY3Q1/2	Modeling language and env (final) Design space	Detection and recovery in Layer 2 of ARMOR; Study DL.
FY3Q3/4	Synthesis to Diagnosability tool Synthesis to performance simulator (final)	Detection and recovery in Layer 3 of ARMOR; Study DL.
FY4Q1/2	Design space (final)	Full scale Runtime Environment test
FY4Q3/4	Synthesis to Reliability tool Synthesis to Diagnosability tool (final)	Large scale evaluation on BTeV hardware and revision
FY5Q1/2	Synthesis to Reliability tool (final)	Final evaluation on BTeV hardware

7. Conclusion

Many real-time embedded systems applications require high computational performance and high availability. High Energy Physics is only one of many disciplines that must collect and analyze huge amounts of data in real time, and must continue operating under fault conditions. The following statement was taken from the President's Information Technology Advisory Committee Report (PITAC)[67]:

> "The Nation needs robust systems, but the software our systems depend on is often fragile. Software fragility is its tendency not to work properly – or at all. Fragility is manifested as unreliability, lack of security, performance lapses, errors, and difficulty in upgrading. Examples can be found everywhere, from our huge information systems for air-traffic control to the personal computers on our desks, from the Pentagon to the Internal Revenue Service (IRS)."

This research proposes to develop new technologies for creating software for real-time embedded computer systems that must exhibit ultra high performance, must be highly available, and must be maintained and evolved easily over the system life-cycle. The results of the research will be a powerful software system that will support the BTeV experiment, as well as new engineering methodologies for developing software for large-scale embedded computer systems. These results could easily be applied to other such applications, including those referred to by PITAC. This research addresses the unreliability, performance lapses, errors, and difficulty of upgrading mentioned there.

This project will bring together a team of experts to develop new embedded systems technologies, which can be made available to a much wider range of applications and researchers. The team will prove the technologies by deploying them in support of the very demanding BTeV trigger and data acquisition system. By collaborating with an ongoing effort in experimental particle physics, the team will have access to a large-scale system for testing without needing to acquire all the hardware independently. Members of the team, which come from universities in the US and from Fermilab, are experienced in the development of this kind of software and will, through the support obtained from this project, make a breakthrough in software for embedded real time systems while at the same time advancing the investigation of a question of major significance in physics.

[1] The BTeV Proposal (May 2000) resides at:
http://www-btev.fnal.gov/public_documents/btev_proposal/index.html.
The BTeV Trigger is described in chapter 9 and the Data Acquisition System in chapter 10. These are located in Part 2 of the document. Trigger algorithm physics simulations are described in chapter 14, located in Part 3.

[2] Fermilab Director Michael Witherell's report on BTeV approval:
http://www-btev.fnal.gov/public_documents/Approval/index.html.

[3] Links to the home pages of members of the collaboration may be found at:
http://www.hep.vanderbilt.edu/btev_rtes/.

[4] Robert Tschirhart and Peter Wilson, private communication.

[5] A general presentation on the fundamental objectives of High Energy Physics in the 21st century can be found at:
http://www.fnal.gov/pub/inquiring/matter/future/index.html.

[6] Schaller, S.C., ed. *1999 IEEE Conference on Real-Time Computer Applications in Nuclear Particle and Plasma Physics*, Santa Fe, June 1999, in IEEE Trans. Nucl. Sci., Vol. 47, Issue 2, Part 1, April 2000.

[7] Gottschalk, E. E., et al., "BTeV Detached Vertex Trigger," to be published in *Proceedings of the 9th International Workshop on Vertex Detectors (Vertex 2000)*, Homestead, MI, September 2000.

[8] A WEB-based animation of the pattern recognition algorithm used in the BTeV trigger can be found at
http://www-btev.fnal.gov/public_documents/animations/Animated_Trigger/index.htm.

[9] A run typically corresponds to all or part of a Tevatron colliding beam "store" which may last several hours. The collision rate decreases during the store as the particles in the beams are gradually used up by the collisions.

[10] Fox, A., Gribble, S., Chawathe, Y., Brewer, E., and Gauthier, P., "Cluster-Based Scalable Network Services," *Symposium on Operating Systems Principles (SOSP-16)*, October 1997.

[11] Bapty, T., Sztipanovits, J., "Model-Based Engineering of Large-Scale Real-Time Systems," *Proceedings of the Engineering of Computer Based Systems (ECBS) Conference*, pp. 467-474, Monterey, CA, March 1997.

[12] Gartner, F., "Fundamentals of fault-tolerant distributed computing in asynchronous environments," *ACM Computing Surveys*, Vol. 31(1), 1999, pp. 1-26.

[13] Kon, F., and Campbell, R., "Supporting automatic configuration of component based distributed systems," *Proceedings of the 5th USENIX Conference on Object-Oriented Technologies & Systems*, 1999.

[14] Bagchi, S., Srinivasan, B., Whisnant, K., Kalbarczyk, Z., Iyer, R.K., "Hierarchical Error Detection in a Software Implemented Fault Tolerance (SIFT) Environment," *IEEE Transactions on Knowledge and Data Engineering*, vol.12, no.2, 2000, pp.203-224.

[15] Sztipanovits, J., et al., "MULTIGRAPH: An Architecture for Model-Integrated Computing," *Proceedings of the IEEE ICECCS'95*, pp. 361-368, November 1995.

[16] Franke, H., Sztipanovits J., and Karsai, G., "Model-Integrated Computing", *Proceedings of the 1997 Hawaii Systems Sciences Conference*, (no page number available, CD-ROM publication), 1997.

[17] Lee, E.A., and Sangiovanni-Vincentelli, A., "A Framework for comparing Models of Computation," *IEEE Transactions on CAD*, Vol. 17, No. 12, December 1998.

[18] Najjar, W., Lee E., and Gao G., "Advances in the dataflow computational model," *Journal of Parallel Computing*, pp. 1907-1929, vol. 25, 1999.

[19] Misra, A., "Sensor-Based Diagnosis of Dynamical Systems," Ph.D. Dissertation, Vanderbilt University, Electrical Engineering, 1994.

[20] Carnes, J., and Misra, A., "Model-Integrated Toolset for Fault Detection, Isolation and Recovery (FDIR)," *International Conference and Workshop on Engineering of Computer Based Systems*, Friedrichshafen, Germany, March 1996.

[21] Harel, D., "Statecharts: A Visual Formalism For Complex Systems," *Science of Computer Programming 8*, pp. 231-278, 1987.

[22] *Object Constraint Language Specification*, Version 1.1, Object Management Group, September 1997.

[23] Neema, S., "System-Level Synthesis of Adaptive Computing Systems," Ph.D. Dissertation, Vanderbilt University, Electrical Engineering, 2001.

[24] Davis, J., Bapty, T., Karsai, G., Malloy, D., Sztipanovits, J., and Tibbals, T., "Model Based Data Validation," *Proceedings of the Joint Technology Showcase on Integrated Monitoring, Diagnostics, and Failure Prevention*, Mobile, AL, April 1996.

[25] Bapty, T., Neema, S., Scott, J., Sztipanovits, J., and Asaad, S., "Model-Integrated Tools for the Design of Dynamically Reconfigurable Systems," *VLSI Design*, 10, 3, pp. 281-306, 2000.

[26] Davis, J., "Integrated Safety, Reliability, and Diagnostics of High Assurance, High Consequence Systems," Ph.D. Dissertation, Vanderbilt University, Electrical Engineering, 2000.

[27] Wilkes M., Lynd L., Sztipanovits J., Karsai G., "The Multigraph Approach to Parallel, Distributed, Structurally Adaptive Signal Processing", *IEEE International Conference on Acousitc and Signal Processing*, pp 2037-2040, 1990.

[28] Bryant, R., "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, pp. 677-691, vol. C-35, no. 8, August 1986.

[29] Bryant, R., "Symbolic Manipulation with Ordered Binary Decision Diagrams," School of Computer Science, Carnegie Mellon University, Technical Report CMU-CS-92-160, July 1992.

[30] Sycara, K., and Liu, J.S., "Multiagent Coordination in Tightly Coupled Task Scheduling," *Proceedings of the First International Conference on Multiagent Systems*, pp. 181-188, 1996.

[31] Rosenschein, J. S., and Zlotkin, G., "Designing Conventions for Automated Negotiation," *AI Magazine*, pp. 29–46, 1994.

[32] Bapty, T., Neema, S., Scott, J., Sztipanovits, J., and Asaad, S. "Model-Integrated Tools for the Design of Dynamically Reconfigurable Systems," ISIS Technical Report/Vanderbilt University, 2000. An online copy is available at: http://www.isis.vanderbilt.edu/publications/archive/Bapty_T_0_0_2000_Model_Inte.PDF.

[33] http://www.isis.vanderbilt.edu/projects/acs/index.html.

[34] Details may be found at http://www.tidsp.com/.

[35] See http://www.ose.com.

[36] see, for example: Bapty T., Abbott B. "Portable Kernel for High-Level Synthesis of Complex DSP-Systems," *Proceedings of the the International Conference on Signal Processing Applications and Technology*, Boston, MA, May 1995, and Montague, B.R., "JN: An Operating System for an Embedded Java Network Computer," Computer Science Technical Report UCSC-CRL-96-29, UCSC, December 1996.

[37] Kalbarczyk, Z., Iyer, R.K., Bagchi, S., and Whisnant, K., "Chameleon: A Software Infrastructure for Adaptive Fault Tolerance," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 10, No. 6, pp. 560-579, June 1999.

[38] Bagchi, S., Srinivasan, B., Whisnant, K., Kalbarczyk, Z., and, Iyer, R.K., "Hierarchical Error Detection in a Software Implemented Fault Tolerance (SIFT) Environment," *IEEE Transactions on Knowledge and Data Engineering*, vol.12, no.2, 2000, pp. 203-224.

[39] Stott, D., Floering, B., Burke, D., Kalbarczyk, Z., and Iyer, R.K., "NFTAPE: A Framework for Assessing Dependability in Distributed Systems with Lightweight Fault Injectors," in *Proc. of 4th Int. Computer Performance and Dependability Symposium IPDS'00*, March 2000, pp. 91-100.

[40] Egan, A., Kutz, D., Mikulin, D., Melhem, R., and Mosse, D., "Fault-Tolerant RT-Mach (FT-RT-Mach) and an Application to Real-Time Train Control", *Software Practice and Experience* (April 1999), Vol. 29 No. 4, pp. 379-395, Wiley Publishers.

[41] A *run* is the overall concept of preparing to and acquiring data for a certain period of time. Run control is the overseeing software that manages a run, which which can range from a few minutes to several hours.

[42] Colajanni, M., Yu. P.S., and Dias, D.M., "Analysis of Task Assignment Policies in Scalable Distributed {Web}-Server Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 6, 1998.

[43] Kuchcinski K., "Embedded System Synthesis by Timing Constraints Solving," *Proceedings of the 10th International Symposium on System Synthesis*, 1997.

[44] Teich J., et al, "An evolutionary approach to system-level synthesis," *Proceedings of the 5th International Workshop on Hardware/Software Co-Design (Codes/CASHE)*, 1997.

[45] Kalavade A., Lee, E., "The Extended Partitioning Problem: Hardware/Software Mapping and Implementation-Bin Selection," *Proceedings of the 6th International Workshop on Rapid Systems Prototyping*, North Carolina, June 1995.

[46] Yokoo, M., Durfee, E.H., Ishida, T., and Kuwabara, K., "Distributed constraint satisfaction for formalizing distributed problem solving," *IEEE 12th International Conference on Distributed Computing Systems*, pp. 614-621, 1992

[47] Parunak, Ward, and Sauter, "The MarCon Algorithm: A Systematic Market Approach to Distributed Constraint Problems," *AI-EDAM* 13 (1999), pp. 217-234. A summary appears in the *Proceedings of ICMAS'98*.

[48] Eiben, A. E., van Hemert, J.I., Marchiori, E., and Steenbeek, A.G., "Solving binary constraint satisfaction problems using evolutionary algorithms with an adaptive fitness function," *Proceedings of the 5th Conference on Parallel Problem Solving from Nature*, pp. 196-205, Berlin, 1998.

[49] Craenen, B.G.W., Eiben, A.E., et al., "Solving Constraint Satisfaction Problems with Heuristic-based Evolutionary Algorithms," *Proc. of the 2000 Congress on Evolutionary Computation*.

[50] Bhattacharya S, Murthy P., Lee E, "Software Sythesis from Datflow Graphs," Kluwer Academic Press, Norwell MA, 1996.
[51] See http://get.qnx.com.
[52] See http://www.wrs.com/.
[53] Birman K.P., and van Renesse, R., "Reliable Distributed Computing with the ISIS Toolkit," IEEE Computer Society Press, Los Alamitos, CA 1994.
[54] van Renesse, R., Birman, K.P., and Maffeis, S., "Horus: A Flexible Group Communication System," *Communications of the ACM*, Vol. 39, No. 4, 1996, pp. 76-83.
[55] Moser, L.E., Melliar-Smith, P.M., Agarwal, D.A., Budhia, R.K., and Lingley-Papadopoulos, C.A., "Totem: A Fault-Tolerant Multicast Group Communication System," *Comm. of the ACM*, vol. 39, No. 4, 1996, pp. 54-63.
[56] Powell, D., "Lessons Learned from Delta-4," *IEEE Micro*, vol. 14, No. 4, 1994, pp. 36-47.
[57] Maffeis, S., "Piranha: A CORBA Tool for High Availability," *IEEE Computer*, vol.30, No.4, 1997, pp. 59-66.
[58] Cukier, M., et al., "AQUA: An Adaptive Architecture that Provides Dependable Distributed Objects," to appear in *Proc. SRDS-17*, 1998, pp.245-253.
[59] Chen, F., Craymer, L., Deifik, J., et al., "Demonstration of the Remote Exploration and Experimentaion (REE) Fault-Tolerant Parallel-Processing Supercomputer for Spacecraft Onboard Scientific Data Processing," *Proc. of International Conference on Dependable Systems and Netwroks*, DSN '00, June 2000, pp. 367-372.
[60] Ferraro, R., "NASA Remote Exploration and Experimentaion Project," http://www-ree.jpl.nasa.gov.
[61] Hojvat, C., private communication. Letter of support associated with this proposal. For details on the Pierre Auger Project, see http://www.auger.org/.
[62] A. Litke, private communication. Letter of support associated with this proposal.
See http://www.snl-e.salk.edu/Technology/.
[63] QuarkNet; see http://quarknet.fnal.gov/.
[64] University of Pittsburgh's Link-to-Learn (http://www.cs.pitt.edu/L2L) and College in High School (http://www.pit.edu/~chsp) programs.
[65] An example of university course initiatives is: PITT CS 3420 Fault Tolerant Parallel and Distributed Systems (http://www.cs.pitt.edu/GrEdu/course-descriptions/3420.html).
[66] Fermitools; see http://www.fnal.gov/fermitools/.
[67] PITAC - Report to the President, President's Information Technology Advisory Committee, 2/24/1999.